Calibration of Machine Learning Models in glmnetr

Walter K. Kremers, Mayo Clinic, Rochester MN

18 August 2025

The Package

The "An Overview of glmnetr" vignette shows how to run the main package function nested.glmnetr() and how to summarize model performances. If one identifies a well performing model according to the metrics in this summary, e.g. concordance, correlation, deviance ratio, linear calibration coefficients, one may want to do further evaluation in terms of calibration. The strongest calibration and validation will involve calibration with new, independent datasets. Frequently one will not have immediate access to such new data sets, or one may want first to do an internal validation before subjecting a model to an external validation. Here we consider an internal validation approach using cross validation or bootstrap re-sampling, similar to how we numerically assessed model performance.

An Example Analysis

To explore calibration we first consider the nested.glmnetr() call from the "An Overview of glmnetr" vignette which fit machine learning models to survival data with family="cox", i.e.

```
set.seed(465783345)
nested.cox.fit = nested.glmnetr(xs, NULL, yt, event, family="cox",
                                dolasso=1, dostep=1, steps_n=40, folds_n=10, track=1)
```

How we gereated the data for ths function call, i. xb matrix as well as the yt and event vectors, is too decribed in the "An Overview of glmnetr" vignette.

Linear Calibration

Using either print() or summary() on the output object nested.cox.fit one gets, amongst other information, summaries for the linear calibration slopes and intercepts as in

```
summary( nested.cox.fit )
```

```
##   Sample information including number of records, events, number of columns in
##   design (predictor, X) matrix, and df (rank) of design matrix:
##              family                    n               nevent
##                 cox                 1000                  698
##          xs.columns                xs.df      null.dev/nevent
##                 100                   94                12.43
## null.m2LogLik/nevent  sat.m2LogLik/nevent
##                12.43                    0
##
##  For LASSO, and Stepwise regression tuned by df and p, average (Ave) model
##  performance measures from the 10-fold (NESTED) Cross Validation are given together
##  with naive summaries calculated using all data without cross validation
##
##                         Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## lasso                       0.2457    1.0907          0.8736         45.6
## lassoR                      0.2473    1.0149          0.8742         20.4
## lassoR0                     0.2455    0.9667          0.8740         14.9
## elastic net                 0.2480    1.0203          0.8746         20.8
## ridge                       0.2340    1.1025          0.8674         99.0
## elastic net G0              0.2448    0.9557          0.8736         16.5
## elastic net G1              0.2457    1.0907          0.8736         45.6
##                         Naive DevRat Naive Concordance Non Zero
## lasso                         0.1683            0.8789       39
## lassoR                        0.1663            0.8759       14
## lassoR0                       0.1663            0.8759       14
## elastic net                   0.1713            0.8790       21
## ridge                         0.1718            0.8822       99
## elastic net G0                0.1663            0.8759       13
## elastic net G1                0.1683            0.8789       37
##
##                         Ave DevRat Ave Slope Ave Concordance Ave Non Zero
## Stepwise df tuned           0.2541    0.9741          0.8776         14.7
## Stepwise p tuned            0.2549    0.9775          0.8786         15.0
##                         Naive DevRat Naive Concordance Non Zero
## Stepwise df tuned             0.1711            0.8785       15
## Stepwise p tuned              0.1711            0.8785       15
```

Here we see that for many of the models the linear calibration slope term is near 1, the ideal for perfect calibration. For the Cox model any intercept term can be absorbed into the baseline survival function and there is no pertinent intercept term for calibration.

At first glance the lwoer deviance ratios for the naive models than for the average deviance ratio for the hold out models looks wrong. The deviance and deviance ratio for the Cox model cannot be calculcated simply by adding the deviance from separate sample observations like for the standard guassian or binomial models. Inspecting the deviance formula ne sees that obervations assocated with events have a smaller impact on deviance when they occur at times when there are more sample elements at risk. In this way the deviance ratio tesnds to be smaller for larger samples form teh same population and the smaller this differnece between the averae and naive deviance ratios is not surprising.

A First Visual

An initial calibration consideration was made in the overview vignette by regressing observed outcomes on the predicteds from the final model based upon the relaxed lasso. This regression was made using splines, in particular the pspline() function from within a coxph() call, as in

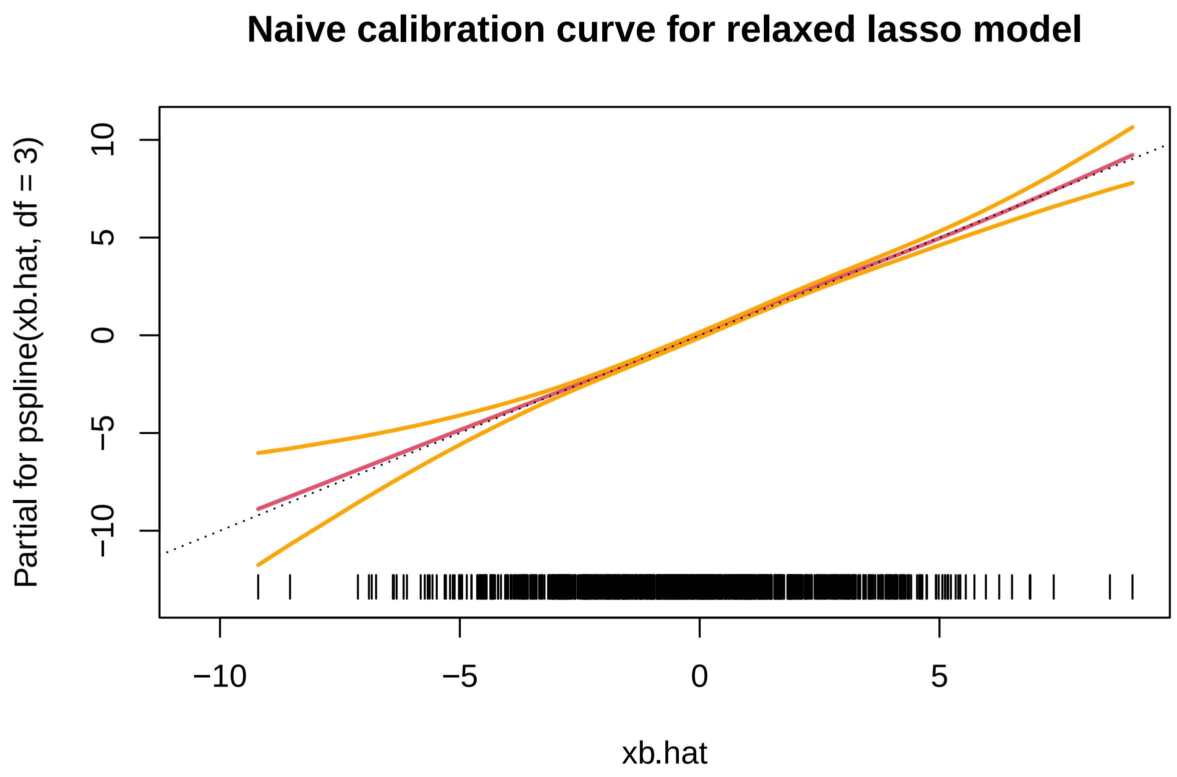
```r
# Get predicteds from CV relaxed lasso model embedded in nested CV output object
xb.hat = predict( object=nested.cox.fit , xs_new=xs, lam=NULL, gam=NULL, comment=FALSE)
# Fit a spline to xb.hat uisng coxph, and plot
#library( survival )                   ## load survival package for Cox model fits
fit1 = coxph(Surv(yt, event) ~ pspline(xb.hat, df=3))
```

```r
summary(fit1)
```

```
## Call:
## coxph(formula = Surv(yt, event) ~ pspline(xb.hat, df = 3))
##
##   n= 1000, number of events= 698
##
##                             coef  se(coef) se2     Chisq   DF   p
## pspline(xb.hat, df = 3),  1.001 0.03121  0.03121 1028.34 1.00 1.2e-225
## pspline(xb.hat, df = 3),                             2.13 2.04  3.5e-01
##
##              exp(coef) exp(-coef) lower .95 upper .95
## ps(xb.hat)3  8.956e+00  1.117e-01 1.494e+00 5.370e+01
## ps(xb.hat)4  8.021e+01  1.247e-02 3.410e+00 1.887e+03
## ps(xb.hat)5  7.170e+02  1.395e-03 1.234e+01 4.165e+04
## ps(xb.hat)6  6.094e+03  1.641e-04 7.332e+01 5.064e+05
## ps(xb.hat)7  5.772e+04  1.732e-05 7.010e+02 4.753e+06
## ps(xb.hat)8  6.805e+05  1.469e-06 8.358e+03 5.541e+07
## ps(xb.hat)9  5.319e+06  1.880e-07 6.447e+04 4.388e+08
## ps(xb.hat)10 5.281e+07  1.894e-08 6.157e+05 4.530e+09
## ps(xb.hat)11 6.531e+08  1.531e-09 6.451e+06 6.613e+10
## ps(xb.hat)12 8.340e+09  1.199e-10 5.227e+07 1.331e+12
##
## Iterations: 4 outer, 14 Newton-Raphson
##      Theta= 0.8077628
## Degrees of freedom for terms= 3
## Concordance= 0.876  (se = 0.005 )
## Likelihood ratio test= 1446  on 3.04 df,   p=<2e-16
```

followed by plotting with

```r
termplot(fit1,term=1,se=TRUE, rug=TRUE, lwd.term=2, lwd.se=2, lty.se=1)
abline(a=0,b=1,lty=3)
title ("Naive calibration curve for relaxed lasso model")
```



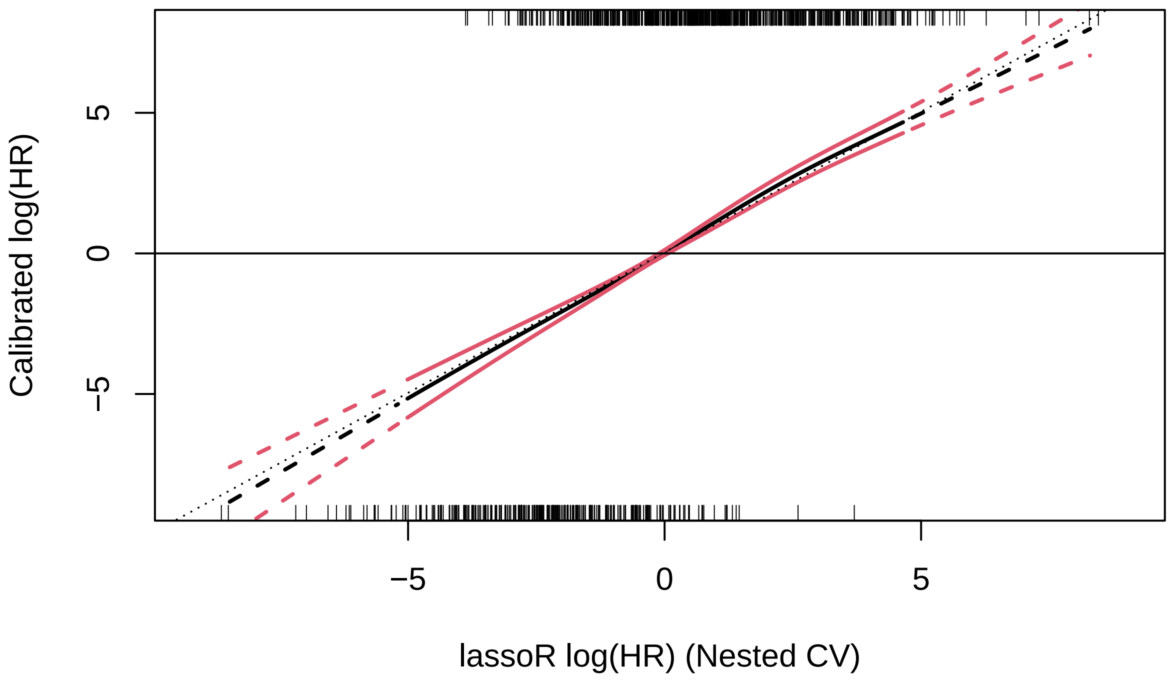
The spline fits may help to understand potential nonlinearities in the model. Here we see, a clibration line which is not far from linear. Still, as noted in the "An Overview of glmnetr" vignette, because the same data are used for model evaluation as well as model derivation, it is hard to put much confidence in such a calibration plot which may suggest a better fit than can be expected for new data.

Calibration Using Spline Fits and Resampling

For each of the models fit, nested.glmnetr() saves the X*Beta's from the model fit using all data. The nested.glmentr() function also calculates the X*Beta's for the hold out data for each partitioning, i.e. each hold out fold of the validation (outer) loop of nested cross validation. In this manner there are multiple model fits and hold out subsets, e.g. k from the k-fold nested CV, for calibration based upon independent observations. Here for each of these subsets we calibrate by regressing outcome on the X*Beta's. While each of these calibrations will individually have limited information, when combined following the principles of cross validation or boostrap sampling, they will collectively provide a more meaningful evaluation. When using the bootstrap option the function also stores the X*Beta's for both the out-of-bag sample elements not selected by the sample with replacement of each bootstrap sample of the validation loop, as well as the in-bag sample elements. A calibration plot can be drawn using the calplot() function as in

```
calplot(nested.cox.fit, wbeta=3)
```

```
##   Range of X*Beta for calibration:
## -8.63979 8.45639
##   Range of calibrated naive confidence intervals:
## -10.9508 9.752533
```

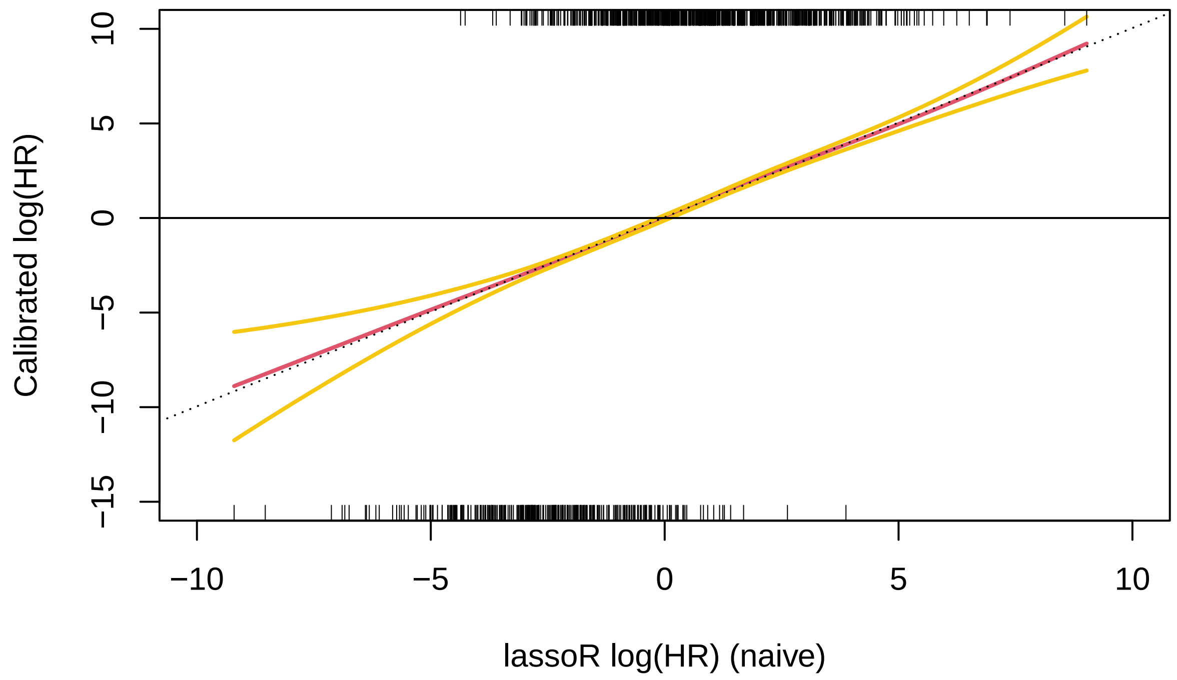


Here we see a smooth, nearly linear predicted log hazard ratio as a function of the model X*Beta from the relaxed lasso model. The bounding lines in red depict the average +/- 2 naive standard errors (SE) to assist in assessing meaningfulness in any deviation from the ideal identity line, and non linearities. In these curves the central region with solid lines denotes the region within the range of all the calibration spline fits, i.e. spline fits from all the different leave-out folds of the CV overlap without extrapolation. The dashed lines depict areas out of range for at least one of the leave out folds. Because spline fits can be rather uncertain when extrapolating beyond the data range, one should be more cautious when drawing conclusions in the dashed regions of these plots. Here we see somewhat wider confidence bounds about the overall calibration curve than in the naive calibratio figure.

Bengio et al. showed that the standard deviations from cross validation might not be accurate for estimation of the actual standard errors. Bates et al. showed that the naive cross validation (CV) estimates and standard errors may be biased, and should thus be viewed with caution. In particular the CV estimates may more closely estimate the expected performance measures over multiple samples than the performance of the model based upon all observed data, and usage the naive CV standard errors, when constructing confidence intervals might be associated with non-coverage three times the nominal non-coverage. This is discussed further in the vignette "An Overview of glmnetr".

A naive calibration curve similar to that shown above (the first figure) can also be easily gotten using the calplot() function when specifying to not use the resample for construction as in

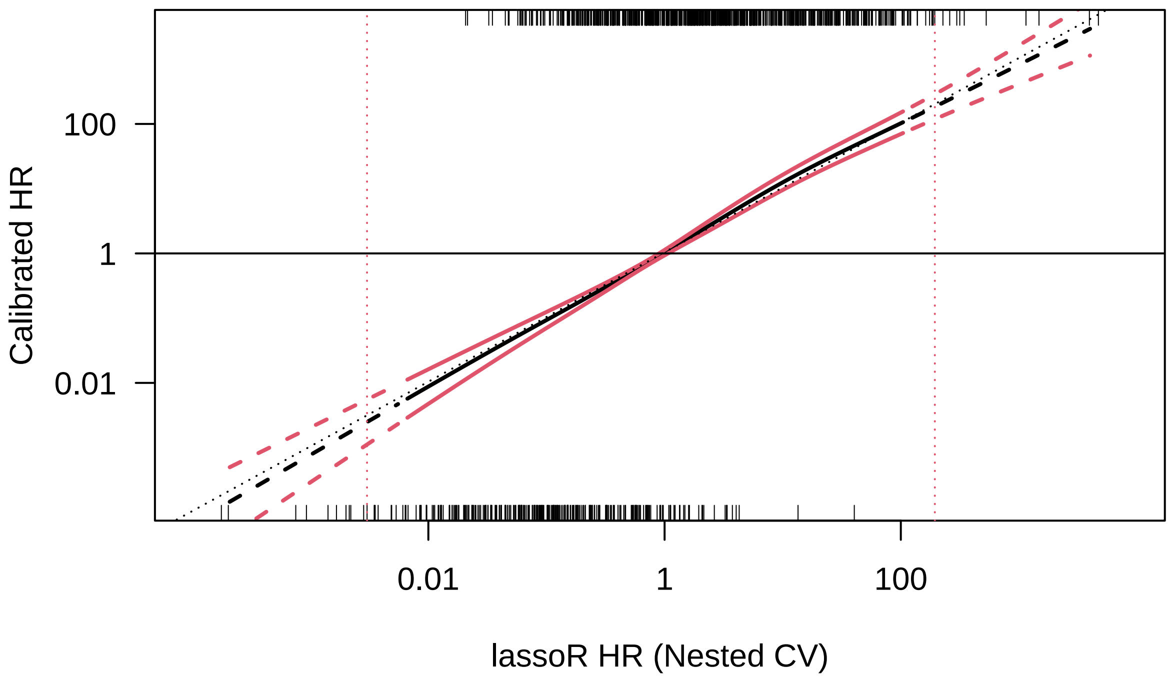
```
calplot(nested.cox.fit, wbeta=3, resample=0, xlim=c(-10,10), ylim=c(-15,10),
        col.term=2, col.se=7)
```



In the calplot() generated figures (so far, for the Cox and binomial models) we see two rugs, one below and one above the plotted region. When using cross validation to evlautae model performance the rug below depicts the model X*Beta's which are not associated with an event and the rug above depicts X*Beta's which are associated with events. When cross validation is used to evaluate model performance the X*Beta are taken from the estimate for the leave out data. When bootstrap is used the X*Beta are either the avarages of the Out-Of-Bag or In-Bag X*Beta's depending on the users input to calplot(). When there are lots of data points it can be hard to read these rugs. One can use the vref option in calplot() to draw two vertical lines where the first separates the smaller vref% of the X*Beta's from the rest, and a second which separates the larger vref% of the data. To depict the hazard ratios (HR) instead of the X*Beta for the Cox model one can use the option plothr, where one assigns a numerical value for the product between tick marks, e.g. exp(1) or 10. Combining these two options we have the example

```
calplot(nested.cox.fit, wbeta=3, vref=1, plothr=100)
```

```
##   Range of X*Beta for calibration:
## -8.63979 8.45639
##   Range of calibrated naive confidence intervals:
## -10.9508 9.752533
```

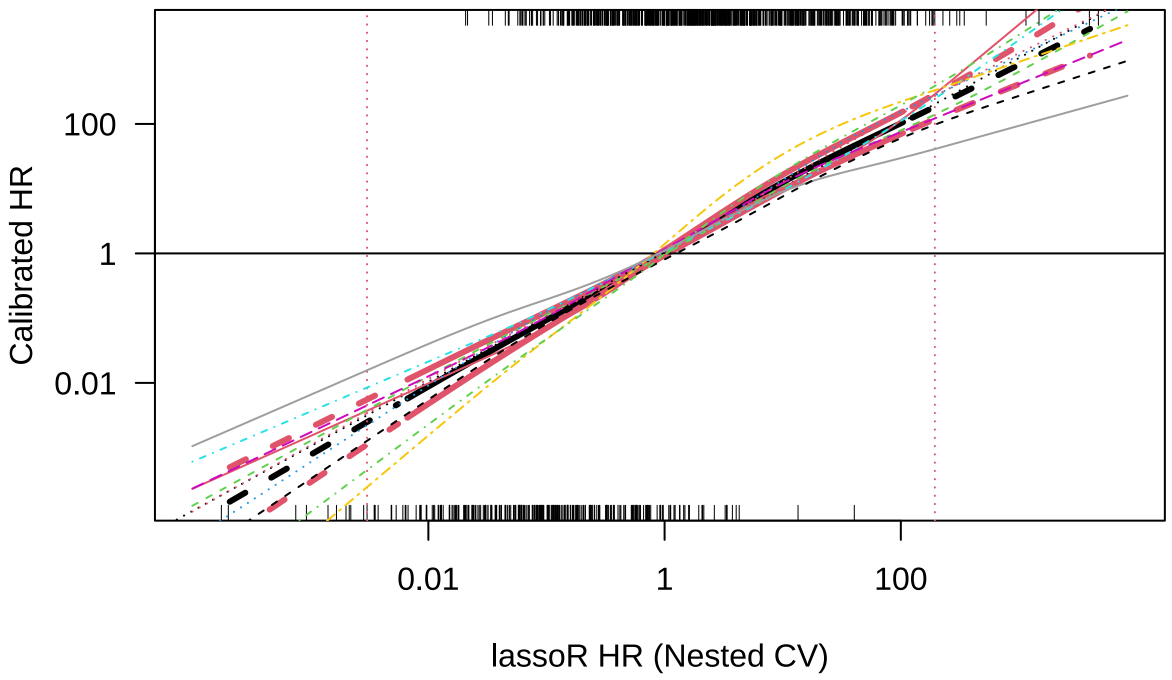


The user can also use different colors for the lines with the options col.term, col.se. One can also specify xlim and ylim in case a few data points cause an excessive amount of white space or odd aspect ratio in the plots.

To view the calibration plots for the individual leave out cross validation folds, one may specify plotfold = 2. In that this generates many figures, we omit in this vignette actually producing plots using this option specification, and instead assign plotfold=1 which overlays the individual calibration curves, albeit without the +/- 2 SE limits for the individual CV folds. The overall calibration (average of the individual CV fold calibrations) and overall +/- 2 SE limits though are maintained.

```
calplot(nested.cox.fit, 3, plotfold=1, vref=1, plothr=100)
```

```
##   Range of X*Beta for calibration:
## -8.63979 8.45639
##   Range of calibrated naive confidence intervals:
## -10.9508 9.752533
```



As we see from the above calls the first term in the calplot() function call is an output object from a nested.glmnetr() call. The second term, wbeta, specifies "which beta" or model is to be used for deriving the model X*Beta's. Here, as we see in the figure x-axis label, the 3 determines the relaxed lasso model. Instead of making a hard to remember key the user can leave this term unspecified and a key will be directed to the R console. The actual numbers for the different models will depend on which models are fit and so this key is dynamic.

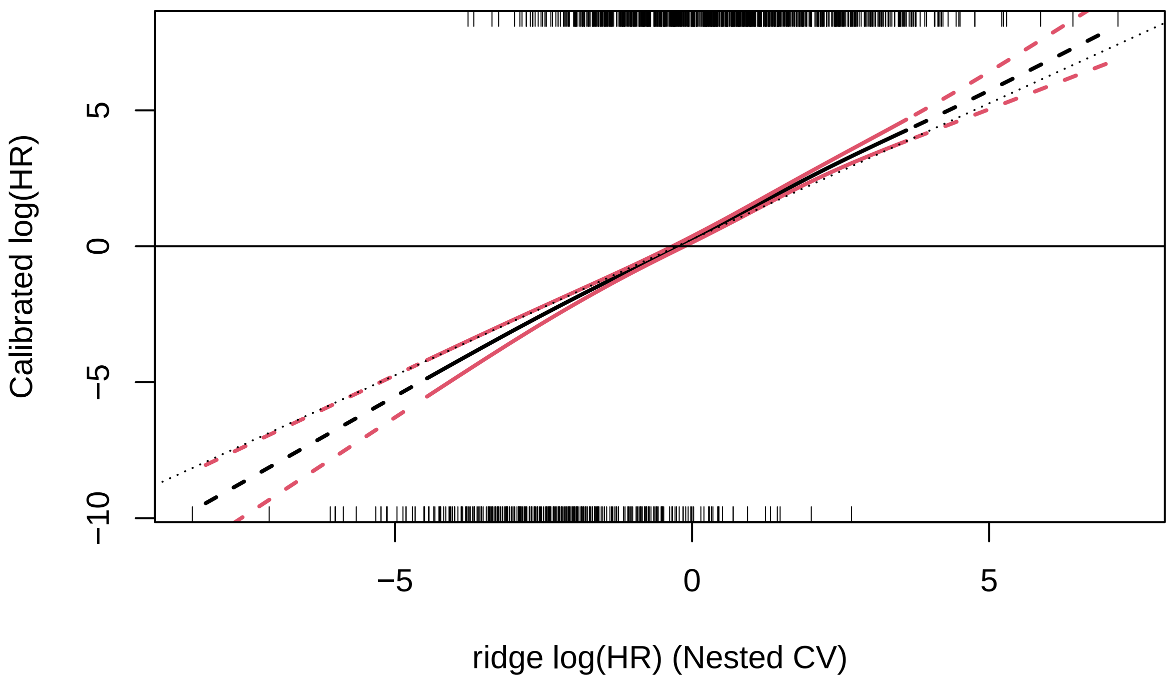
```
calplot(nested.cox.fit)
```

```
##  specify num for wbeta =
##                       Var wbeta
## null           0.000000     1
## lasso          5.244852     2
## lassoR         6.081171     3
## lassoR0        6.691563     4
## elastic        6.027242     5
## ridge          4.620162     6
## elastic G0     6.792812     7
## elastic G1     5.244852     8
## lasso cal      7.287986     9
## lassoR cal     7.040716    10
## lassoR0 cal    6.706474    11
## elastic cal    7.051174    12
## ridge cal      7.897443    13
## elastic G0 cal 6.808074    14
## elastic G1 cal 7.287986    15
## step.df        7.438856    16
## step.p         7.466566    17
```

From this key we read of the numbers corresponding to the respective models. The variance in X*beta for the "null" model is 0 as the intercept for the Cox model is arbitrarily assinged the value of 0 for each resample model fit. From this key we see we can produce a calibration plot for the ridge regression model by setting wbeta = 6 (wbeta for "which beta"), as in

```
calplot(nested.cox.fit, 6)
```

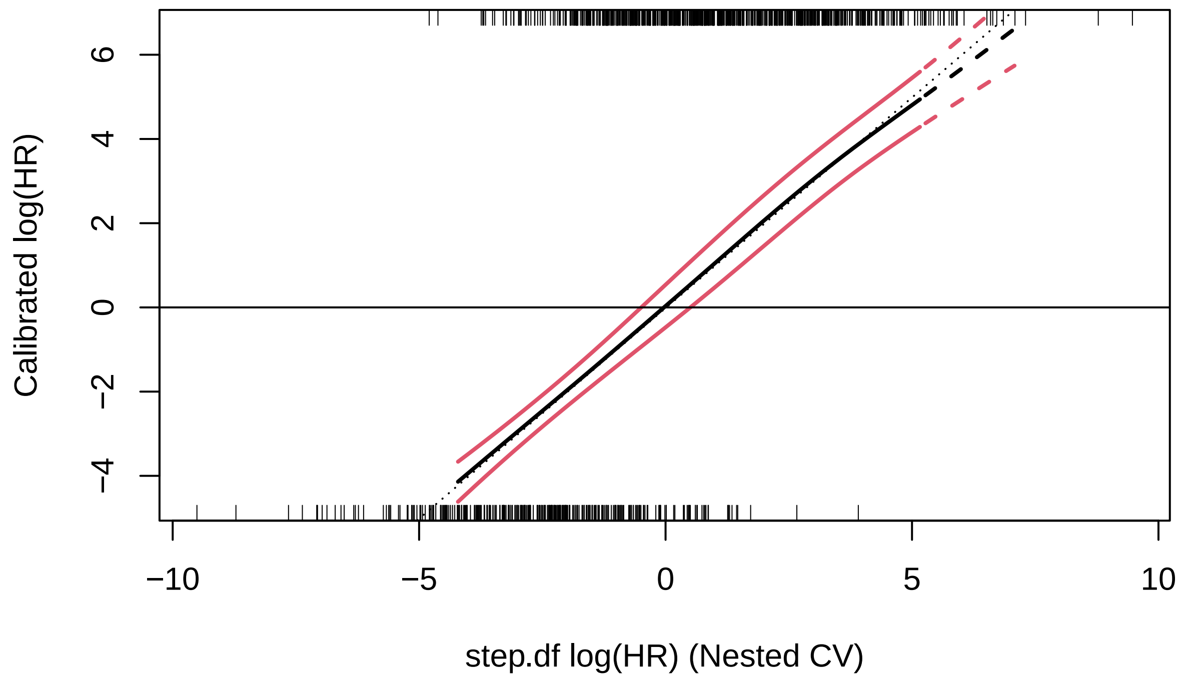
```
##   Range of X*Beta for calibration:
## -8.412004 7.169039
##   Range of calibrated naive confidence intervals:
## -10.85263 9.590673
```



Here we see the model is not ideally calibrated as the calibration curve largely does not include the identity line over the full range of predicteds, and it requires a correction to achieve an un (less) biased estimation of the hazard ratio. Inspecting the calibration curve for a step wise regression model, where the number of terms included in the model is informed by cross validation

```
calplot(nested.cox.fit, 16)
```

```
##   Range of X*Beta for calibration:
## -9.50801 9.472167
##   Range of calibrated naive confidence intervals:
## -4.615066 7.489624
```



we see that the response deviates some form the identity line but not significantly so.

To obtain the numerical values used to construct these calibration plots one may specify plot=0 (or plot=2 to plot and obtain the numerical data) in list format as in

```
tmp = calplot(nested.cox.fit, 3, plot=0)
```

```
##   Range of X*Beta for calibration:
## -8.63979 8.45639
##   Range of calibrated naive confidence intervals:
## -10.9508 9.752533
```

```
str(tmp)
```

```
## List of 5
##  $ estimates     : num [1:101, 1:5] -9.21 -9.02 -8.84 -8.66 -8.48 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:101] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:5] "plotxb" "est" "se" "lower" ...
##  $ est.resample  : num [1:10, 1:101] -8.37 -8.98 -10.16 -7.41 -8.37 ...
##  $ se.resample   : num [1:10, 1:101] 6.12 4.43 4.91 3.99 5.07 ...
##  $ lower.resample: num [1:10, 1:101] -20.6 -17.8 -20 -15.4 -18.5 ...
##  $ upper.resample: num [1:10, 1:101] 3.861 -0.119 -0.329 0.569 1.765 ...
```

These data may be further processed by the user.

A Binomial Model

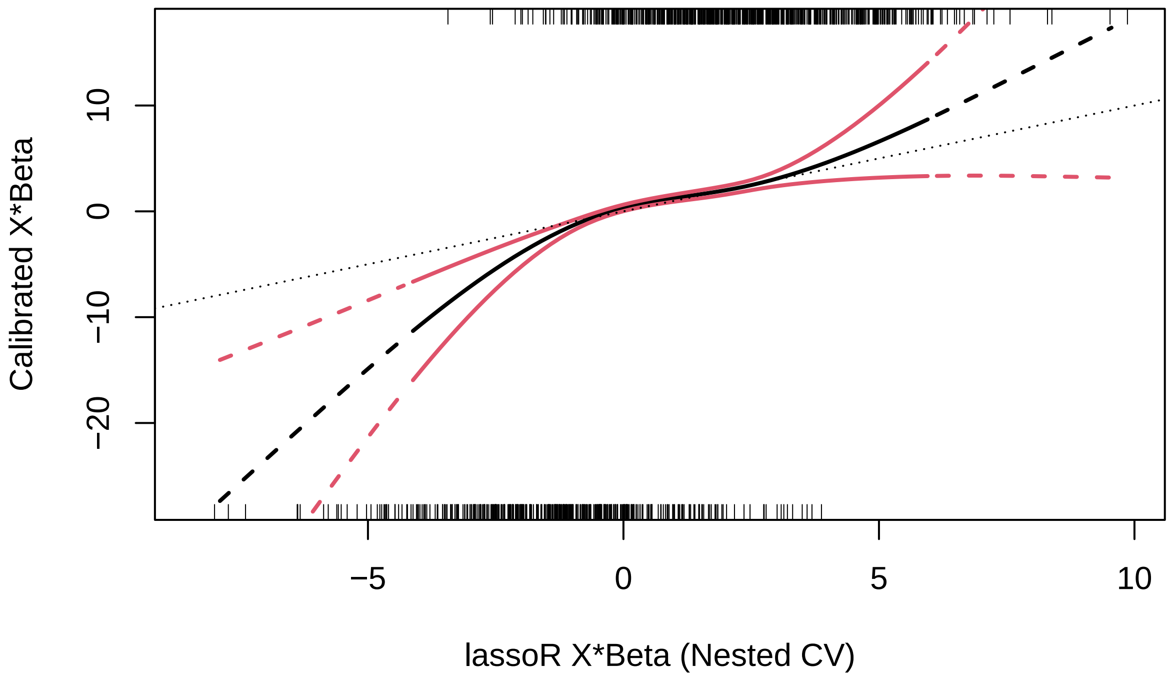
For nested.glmnetr() analyses with family = “binomial” with the call

```
yb = simdata$yb
nested.bin.fit = nested.glmnetr(xs,NULL,yb,NULL,family="binomial",
      dolasso=1, doxgb=list(nrounds=250), dorf=1, doorf=1, doann=1,
      folds_n=10, seed=219301029, track=1)
```

an example calibration plot is

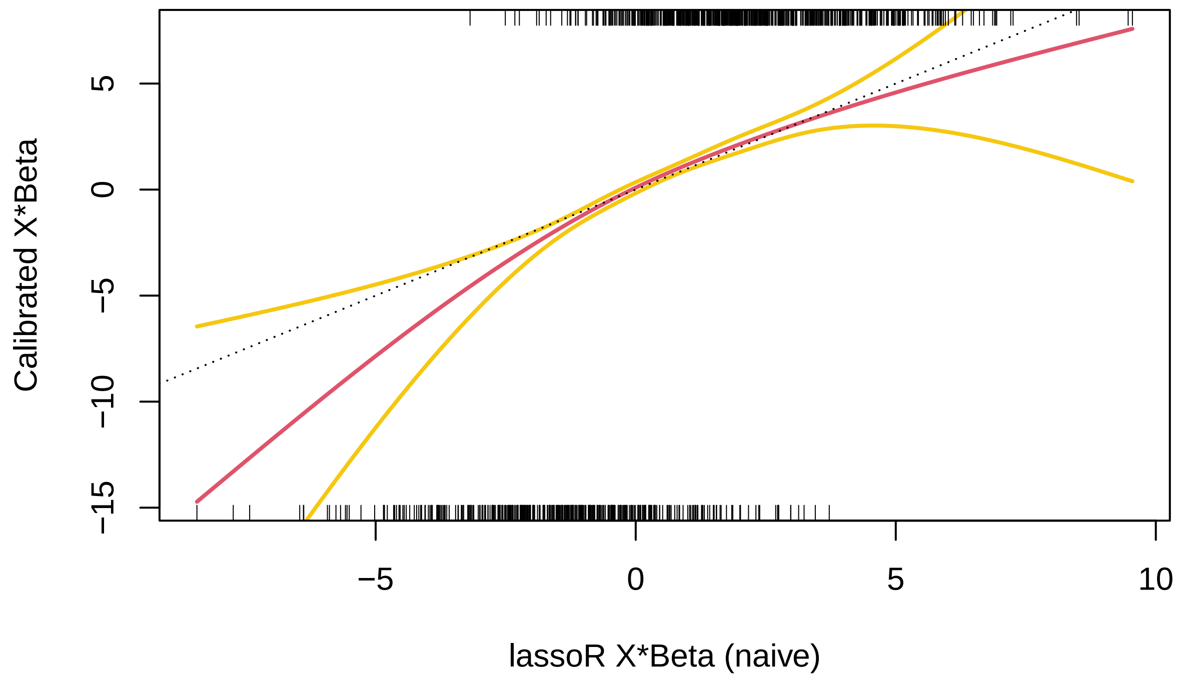
```
calplot(nested.bin.fit, 3, plotfold=0)
```

```
##   Range of X*Beta for calibration:
## -8.002414 9.862854
##   Range of calibrated naive confidence intervals:
## -44.38544 31.51217
```



Since these data were generated with probabilities exp(X*Beta)/(1+exp(X*Beta)) we want that the medal would calibrate linearly. The wide variability for the larger X*Beta, positivly or negatively, is likely due to the lack of stability in estiatmes for probabilities near 0 and 1. Examination of the naive calibration plot which uses the same data for model derivation and calibration

```
calplot(nested.bin.fit, 3, resample=0, df=3, col.term=2, col.se=7)
```



shows a self consistency better than the above plots. While this may calibrate slightly better it is not as linear as we might expect.

A Binomial Model Calibrated Using Bootstrap

Considering this we next evaluate model performances using bootstrap, that is fit models based upon random samples from the original sample (with replacement) of same sample size as the original sample. Then we fit calibration curves for the out-of-bag sample units for each bootstrap sample fitted model, that is the elements of the original sample that are not selected by the bootstrap sample. The number of bootstrap samples for calculation is specified using the *bootstrap* option in the nested.glmnetr() call,

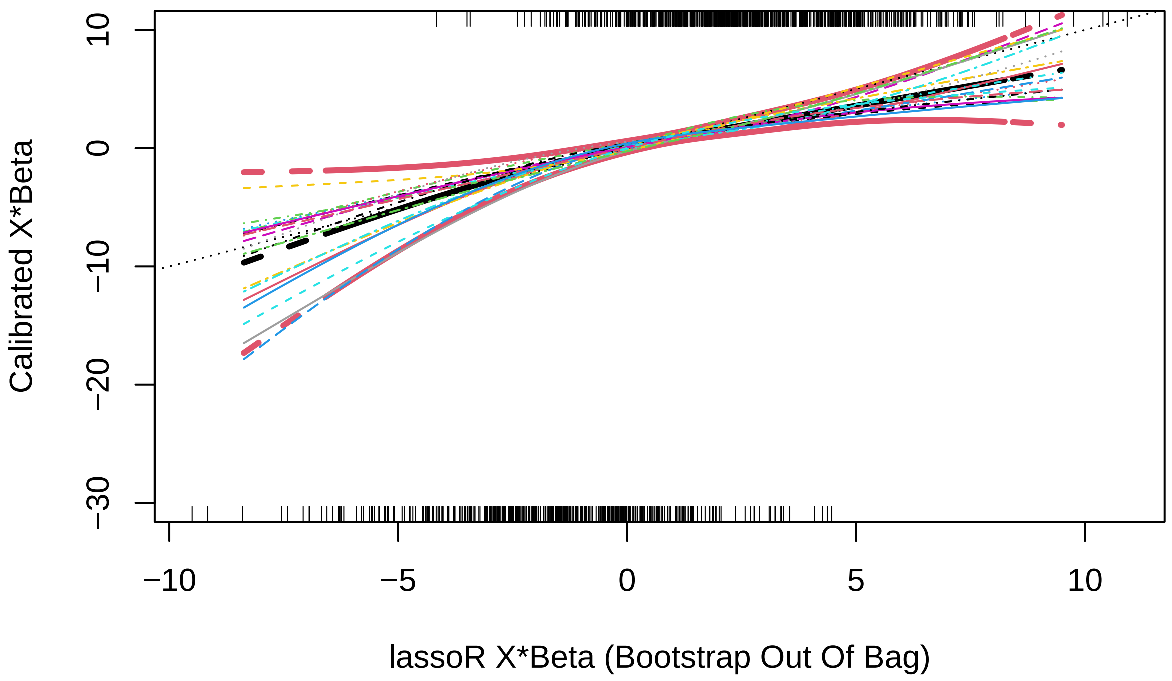
```
yb = simdata$yb
nested.bin.boot.fit = nested.glmnetr(xs,NULL,yb,NULL,family="binomial",
      dolasso=1, dorf=1,
      folds_n=10, seed=219301029, track=1, bootstrap=20)
```

Bootstrap Out-Of-Bag (OOB) Calibrations

The Out-Of-Bag (OOB) calibration plots can then be constructed using the calplot() function as before,

```
calplot(nested.bin.boot.fit, wbeta=3, plotfold=1, ylim=c(-30,10))
```

```
##   Range of X*Beta for calibration:
## -9.501879 10.92212
##   Range of calibrated naive confidence intervals:
## -17.31873 11.26827
```



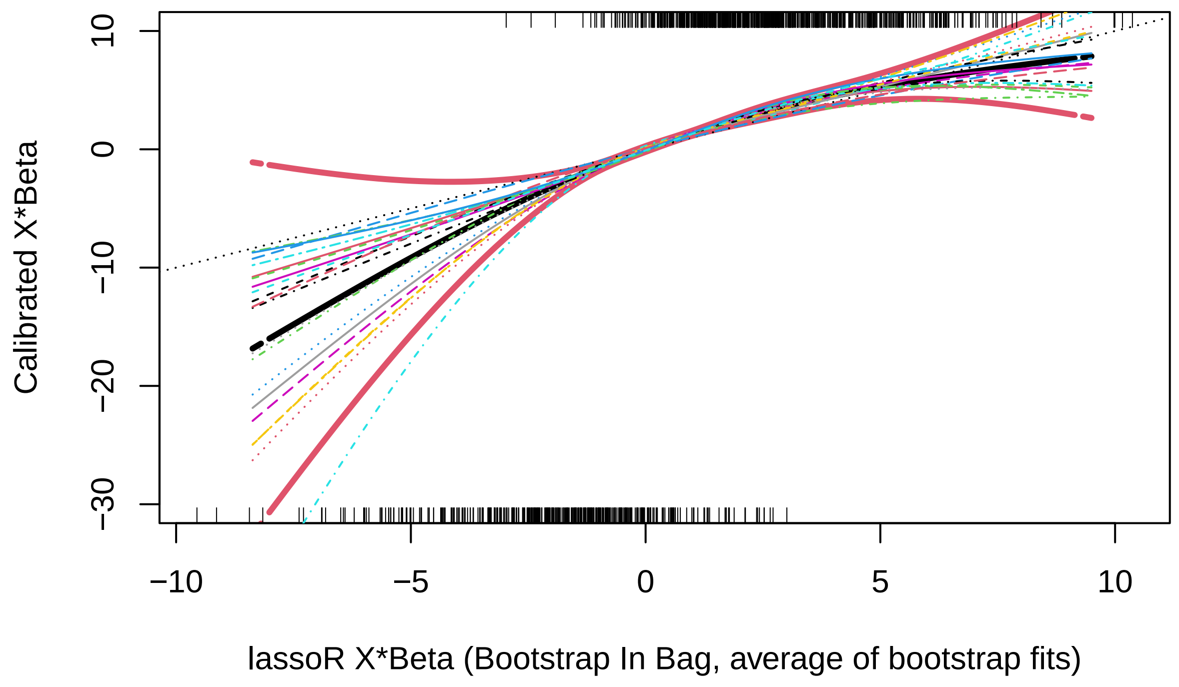
As for the nested CV, the solid center black line is the average of the calibration lines from the mutiple bootstrap resamples. The upper and lower thick red lines are the average +/- 2 standard deviations from the bootstrap resample calibrations. Here we see a calibration plots that seems more on target than we did for the nested cross-validation plots. This may be due to the larger sample size for the hold out datasets in the individual validations, circa 37% of the original sample as opposed to 10% for 10-fold cross-validation. For the cross validation one could consider 3-fold cross validation to increase the hold-out sample sizes, repeat the whole process many times using different seeds (see the glmnet package vignettes) and then average, but we have not done this here. For the bootstrap one can also easily run a larger number of re-samples to improve estimate stability. One might also consider a method similar to the bootstrap which selects unique observations in the resamples and then evaluate on the hold out data for each resample. This can be done using the unique option in nested.glmnetr(), e.g. setting unique=0.63 to randomly select 63% of the sample for model fitting leaving 37% for model evaluation in each resample.

Bootstrap In-Bag Calibrations

In earlier works Austin et al. as well as Riley et al. fit models based upon bootstrap samples and then fit calibration curves based upon the in-bag data points and model X*Beta (predicteds). This mimics the usual procedure for bootstrap analysis. This can be done using the calplot() function by setting the *oob* option to 0,

```
calplot(nested.bin.boot.fit, wbeta=3, oob=0, plotfold=1, ylim=c(-30,10))
```

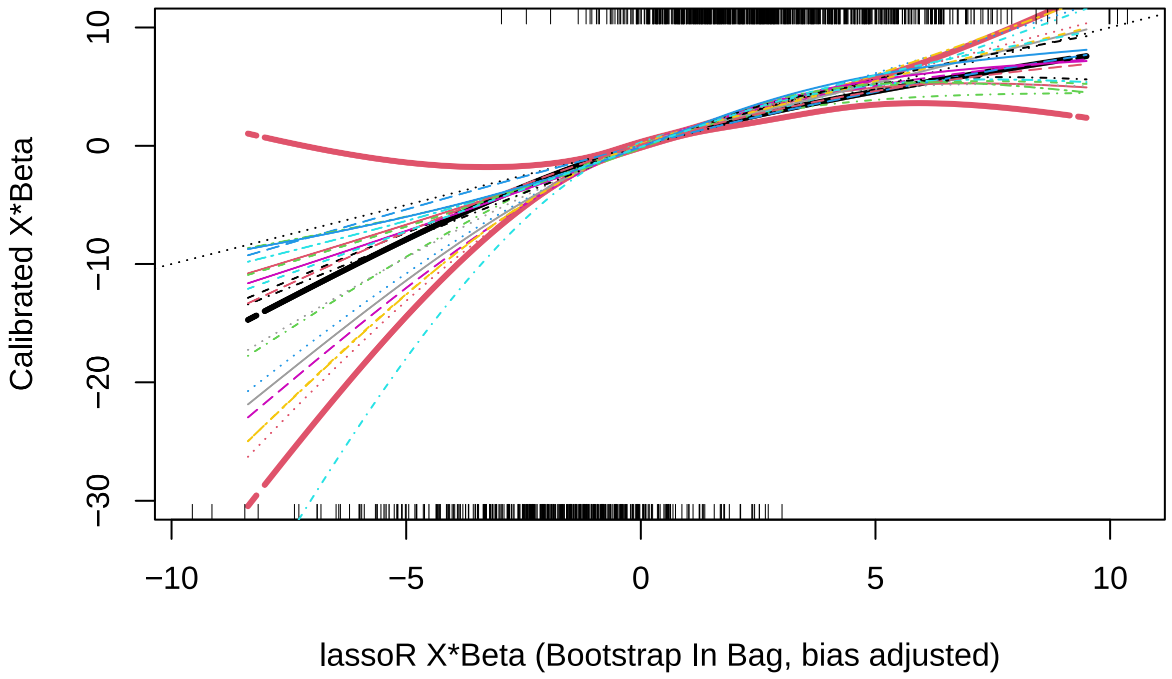
```
##   Range of X*Beta for calibration:
## -9.55692 10.36934
##   Range of calibrated naive confidence intervals:
## -32.58403 13.06498
```



Here the thin lines are taken form in-bag calibration plots from the individual bootstrap resamples. The center thick line is the average of the calibration plots from the individual bootstrap resamples, and the upper and lower thick red lines are the average +/- 2 standard deviations from the bootstrap resample calibrations. Typically when using the bootstrap one common appraoch for estimation is to take the estimate form the full sample analysis and describe confidence interlas interms of this estiamte and the varaiblity in the in-bag resample estimates. This may be gotton using the bootci=1 option as in

```
calplot(nested.bin.boot.fit, wbeta=3, oob=0, bootci=1, plotfold=1, ylim=c(-30,10))
```

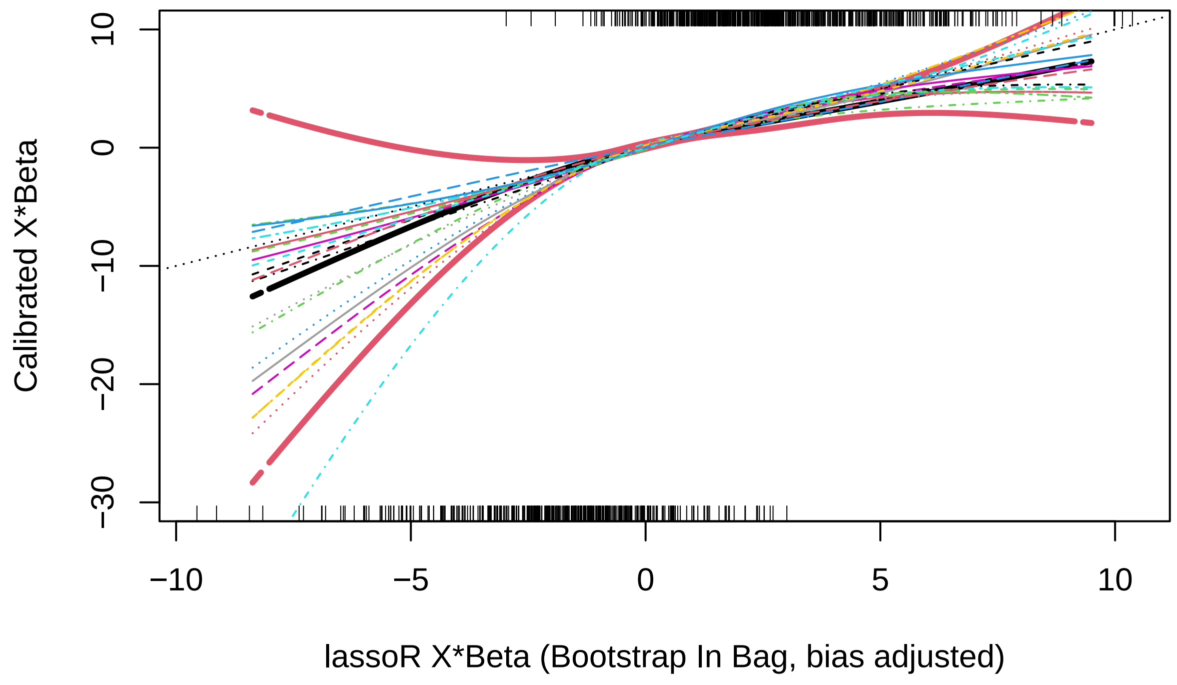
```
##   Range of X*Beta for calibration:
## -9.55692 10.36934
##   Range of calibrated naive confidence intervals:
## -30.45554 12.78563
```



where we see a shift and slight change in shapes of the curves. The bootstrap estiamte can also be bias in the full data model by (Ave(XBeta_i) - XBeta_full). We can make this bias adjustment using the biasbc=1 (bc for bias adjsutment) as with

```
calplot(nested.bin.boot.fit,wbeta=3,oob=0,bootci=1,bootbc=1,plotfold=1,ylim=c(-30,10))
```

```
##   Range of X*Beta for calibration:
## -9.55692 10.36934
##   Range of calibrated naive confidence intervals:
## -28.32705 12.50627
```



This figure again differs slightly form teh above figures. The in-bag bootstrap calibrations here seem different form the out-of-bag calibrations.

Bootstrap Calibration for a Random Forest

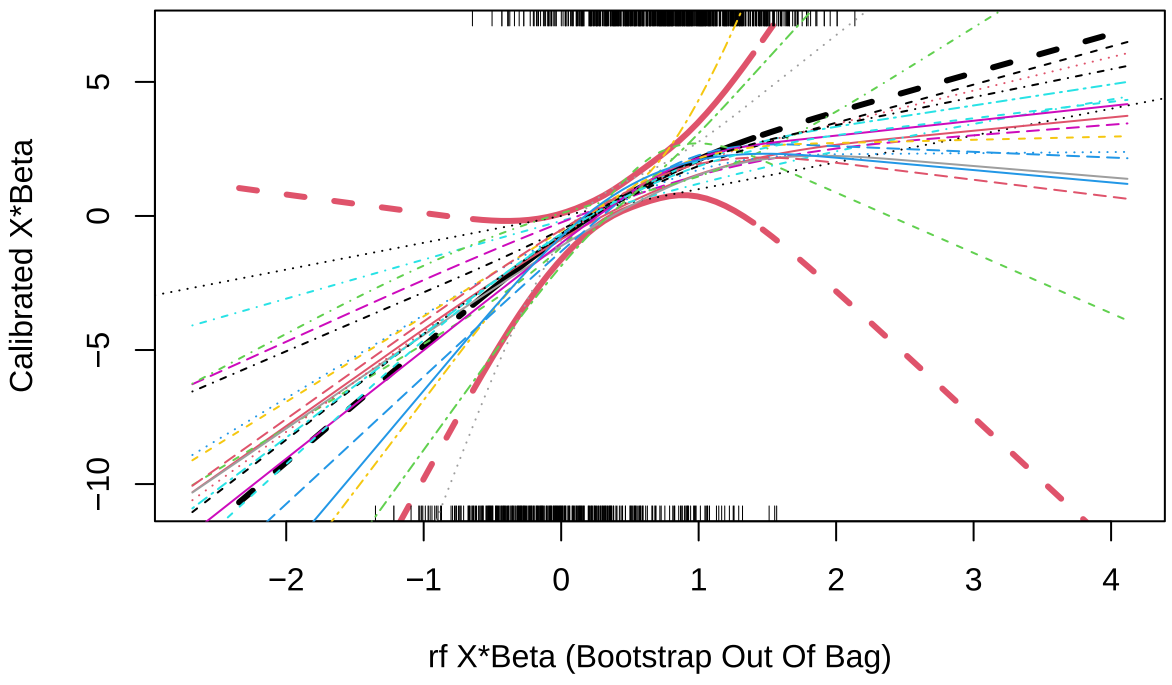
These bootstrap plots above were for the relaxed lasso model fit. We now examine similar bootstrap calibration plots but for random forest models which have greater potential for overfit.

Bootstrap Out-Of-Bag (OOB) Calibrations for a Random Forest

For the out-of-bag calibration bootstrap plots

```r
calplot(nested.bin.boot.fit, wbeta=9, plotfold=1)
```

```
##   Range of X*Beta for calibration:
## -1.350711 2.136162
##   Range of calibrated naive confidence intervals:
## -25.60739 26.75815
```



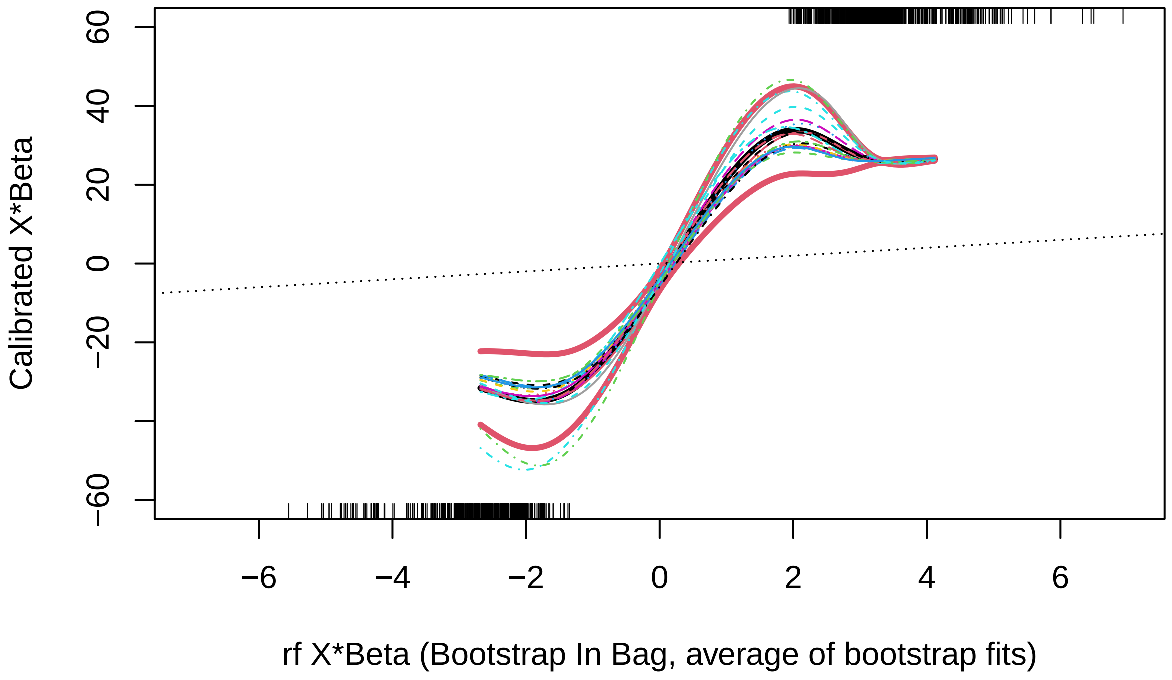
we see that these are highly variable yet consistent with the ideal calibration curve, the identity line.

Bootstrap In-Bag Calibration for a Random Forest

The variability in the in-bag calibratin curves from in-bag bootstrap resamples and for the random forest model is depicted in the graph

```
calplot(nested.bin.boot.fit, wbeta=9, oob=0, plotfold=1, df=6,
        ylim=c(-60,60), xlim=c(-7,7))
```

```
##   Range of X*Beta for calibration:
## -5.552771 6.93748
##   Range of calibrated naive confidence intervals:
## -46.82076 44.97234
```



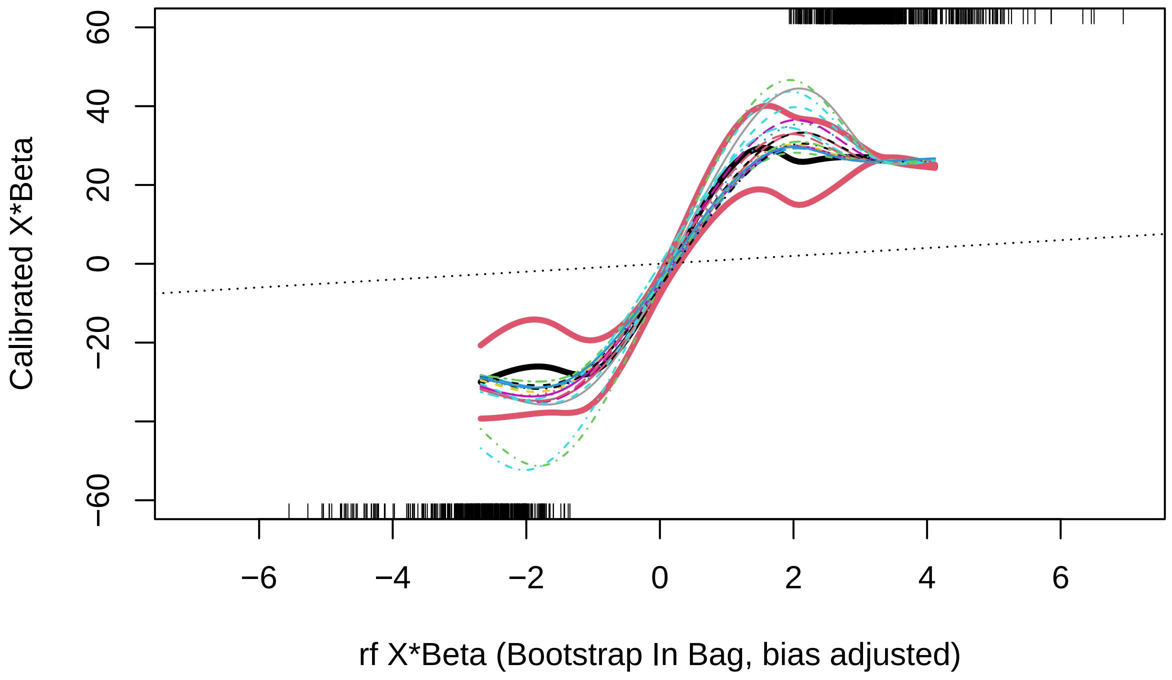
Note, here we used the df=6 option for the degress of freedom in spline fitting to get a set of more reasonable fits.

Bootstrap In-Bag Calibration for a Random Forest using a bootstrap approach

As described above a common way to use the bootstrap to construc confidence intervals is to take the estiame based upon all data and use the bootstrap in-bag sample estimates to inform about the varaiblity of the full sample variblity. This is depeicted in the graph

```r
calplot(nested.bin.boot.fit, wbeta=9, oob=0, bootci=1, plotfold=1, df=6,
        ylim=c(-60,60), xlim=c(-7,7))
```

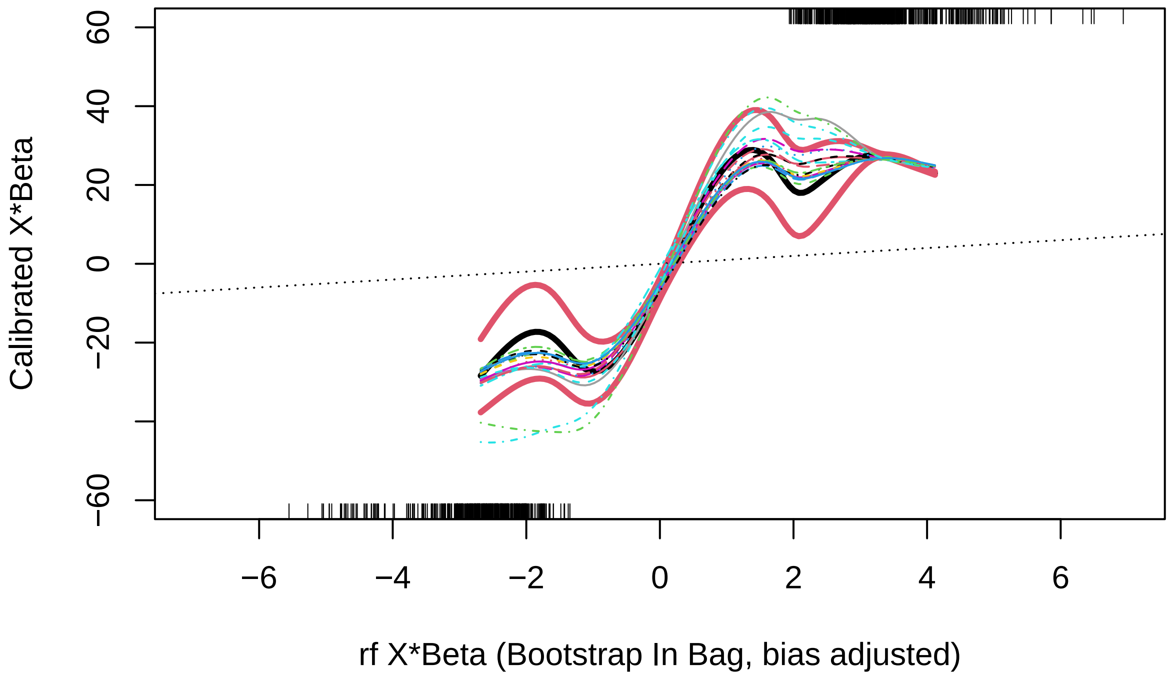
```
##   Range of X*Beta for calibration:
## -5.552771 6.93748
##   Range of calibrated naive confidence intervals:
## -39.28831 40.1166
```



Here we see a strong deviation from the identity line. The tick marks for the rugs (based upon average X*Bata's over the many bootstrap fits) also show that the random forest may largely be separating events from non-events for the logistic model framework, suggesting overfit. The bias corrected bootstrap calibration curves are depecited by

```r
calplot(nested.bin.boot.fit, wbeta=9, oob=0, bootci=1, bootbc=1, plotfold=1, df=6,
        ylim=c(-60,60), xlim=c(-7,7))
```

```
##   Range of X*Beta for calibration:
## -5.552771 6.93748
##   Range of calibrated naive confidence intervals:
## -37.70612 39.00131
```



have a rather unusual and unexpected deviation from the ideal calibration curve of the identity line. This too may be due to overfit.

While the two in-bag calibration curve sets strongly depart from the ideal of the identity line, the out-of-bag curves while possibly highly variable were consistent as a group with the ideal of the identity line. The bootstrap out-of-bag calibration curves have a clearer basis and depict more reasonable findings. In a sense this is not unexpected. When fitting machine learning models one typically evaluates model fit based upon the out-of-bag data points and not the in-bag data points. A difficulty here could be that the calibration curves, calculated at particular values for X*Beta, do not describe a well defined parameter in terms of the distribution function. Potentially as well the fitting process for machine learning models might not lend it self to the assumptions typically used to support bootstrap inferences. In particular as we see here when fitting random forest models, with the multiple repeat observations in the bootstrap resamples the model refits are overly optimistic, more strongly overfit and give more extreme X*Beta as evidenced in these figures. We present the in-bag and biased adjusted bootstrap calibration curves not to promote their usage but to 1) show the variability in model fit and 2) how they may give poor inferences for a simple dataset. The user may want to generate other datasets of different known form and see how the bootstrap performs for their data.

A Normal (Gaussian Errors) Model

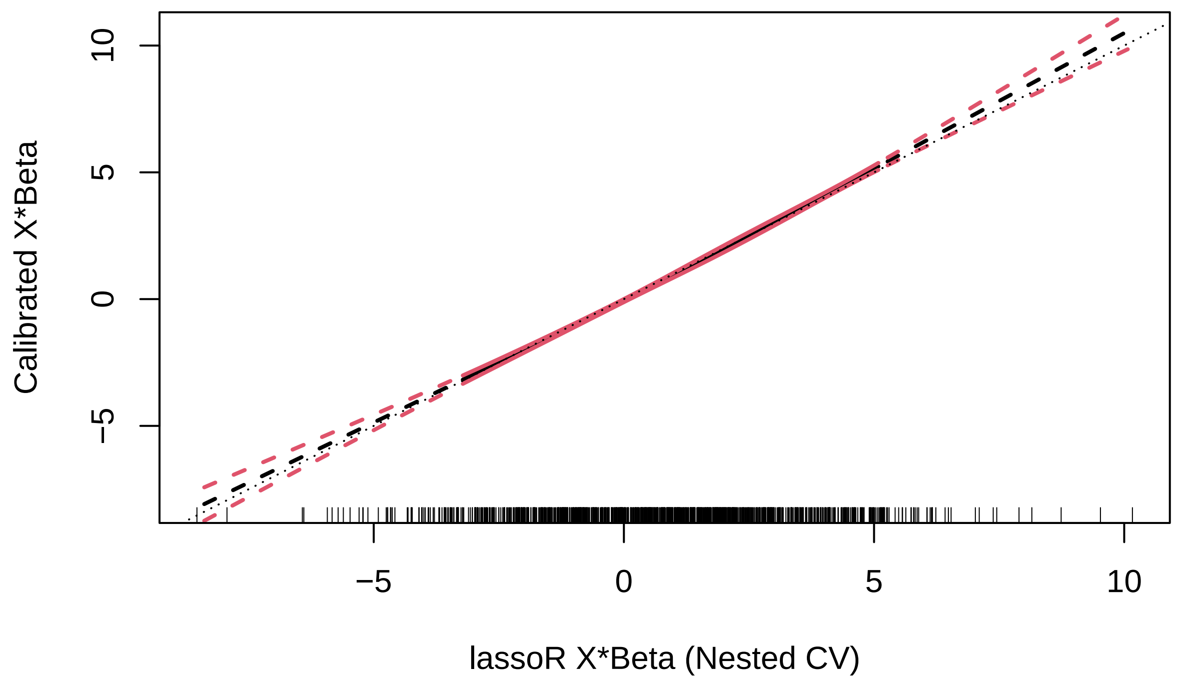
First calculating the numerical summaries of prediction performance

```
nested.gau.fit = nested.glmnetr(xs,NULL,y_,NULL,family="gaussian",
      dolasso=1, doxgb=list(nrounds=250), dorf=1, doorf=1, doann=list(bestof=10),
      folds_n=10, seed=219301029, track=1)
```

and then plotting

```
calplot(nested.gau.fit, wbeta=3)
```

```
##   Range of X*Beta for calibration:
## -8.533496 10.1643
##   Range of calibrated naive confidence intervals:
## -8.744553 11.28182
```



we see a small but probably not significant deviation from the ideal calibration line, the identity function.

Perspective

The summary and calibration plot functions used here do not address all needed for model validation and calibration but do allow a meaningful and un (or minimally) biased summary of model fits. The original outcome variable and X*betas are stored as vectors or matrices in the output with names y_, xbetas (full model) and xbetas.cv (cross-validation) and xbetas.boot.oob and betas.boot.inb (bootstrap out-of-bag and in-bag) allowing the user to further inspect model fits and to perform other means of calibration. For cross-validation analyses the fold information is contained in the output object in the vector foldid. For bootstrap analyses the first column of xbetas.boot.oob and betas.boot.inb describe the replication of the bootstrap sampling process and the second columns the sequential index for the data point in the input dataset.

References

Austin PC, Steyerberg EW. Graphical assessment of internal and external calibration of logistic regression models by using loess smoothers. Stat Med. 2014; 33(3):517-35. doi: 10.1002/sim.5941 .

Bates S, Hastie T, Tibshirani R, “Cross-validation: what does it estimate and how well does it do it?”, 2022, https://arxiv.org/abs/2104.00673 .

Bengio Y & Grandvalet Y, “No Unbiased Estimator of the Variance of K-Fold Cross-Validation”, Journal of Machine Learning Research 5, 2004, 1089–1105, https://www.jmlr.org/papers/volume5/grandvalet04a/grandvalet04a.pdf .

Riley RD, Pate A, Dhiman P, Archer L, Martin GP, Collins GS. Clinical prediction models and the multiverse of madness. BMC Med. 2023; 21(1):502. doi: 10.1186/s12916-023-03212-y .